Symmetry Detection in ROBDDs

Neil Kettle and Andy King

***Abstract*— Detecting symmetries has many applications in logic synthesis that include, amongst other things, technology mapping, deciding equivalence of Boolean functions when the input correspondence is unknown and finding support-reducing bound sets. Mishchenko showed how to efficiently detect symmetries in ROBDDs without the need for checking equivalence of all co-factor pairs. This work resulted in practical algorithms for detecting classical and generalized symmetries. Both the classical and generalized symmetry detection algorithms are monolithic in the sense that they only return a meaningful answer when they are left to run to completion. In this paper we present anytime algorithms for detecting both classical and generalized symmetries, that output pairs of symmetric variables until a prescribed time bound is exceeded. These anytime algorithms are complete in that given sufficient time they are guaranteed to find all symmetric pairs. Anytime generality is not gained at the expense of efficiency since this approach requires only very modest data structure support and offers unique opportunities for optimization so the resulting algorithms are competitive with their monolithic counterparts.**

***Index Terms*— Logic Synthesis, ROBDDs, Symmetry**

I. Introduction

SYMMETRY detection has been important since the days of Shannon [1] who observed that symmetric functions have efficient switch network implementations. Symmetry detection is no less important today and knowledge of symmetric variables has applications in logic synthesis [2], [3], technology mapping [4], [5], combining technology-independent and technology-dependant stages of logic synthesis [6], detecting support-reducing bound sets [7], ROBDD minimization [8], [9] and detecting equivalence of Boolean functions when the input correspondence is unknown [10]–[12].

The challenge in symmetry detection is to find efficient algorithms for detecting all symmetric variables pairs (x_i, x_j) of a given Boolean function $f(x_1 \ldots x_n)$, that is, find all pairs (x_i, x_j) such that $f(x_0, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_0, \ldots, x_j, \ldots, x_i, \ldots, x_n)$. The intuition being that f remains unchanged under the switching of the variables x_i and x_j. This symmetry is formally known as the first-order classical symmetry, or the non-skew non-equivalence symmetry [13]. It can be shown from Boole's expansion theorem [14] that this is equivalent to checking equality of the co-factor pair $f|_{x_i \leftarrow 0, x_j \leftarrow 1} = f|_{x_i \leftarrow 1, x_j \leftarrow 0}$ where $f|_{x_i \leftarrow a, x_j \leftarrow b} = f(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_{j-1}, b, x_{j+1}, \ldots, x_n)$. This notion of symmetry had been generalized [13], [15] to the symmetry types listed in Table I where $f|_{a,b}$ abbreviates $f|_{x_i \leftarrow a, x_j \leftarrow b}$. These symmetries can be categorized into two types depending on whether or not a negated co-factor occurs in the relationship: $T_1, \ldots, T_6$ coincide with those of Zhang *et al.* [12] whereas $T_7, \ldots, T_{12}$ correspond to the $\neg T_1, \ldots, \neg T_6$ types in the notation of Zhang *et al.*

The authors are members of the Computing Laboratory, University of Kent, UK, whose work was funded by the EPSRC Grant EP/C015517 and the British Council Grant PN 05.021.

I: Generalized Symmetry Types

Positive Co-factor relations	**Negative Co-factor relations**
$T_1^{x_i,x_j}(f) \iff f\|_{1,0} = f\|_{0,1}$	$T_7^{x_i,x_j}(f) \iff f\|_{1,0} = \neg f\|_{0,1}$
$T_2^{x_i,x_j}(f) \iff f\|_{0,0} = f\|_{1,1}$	$T_8^{x_i,x_j}(f) \iff f\|_{0,0} = \neg f\|_{1,1}$
$T_3^{x_i,x_j}(f) \iff f\|_{0,0} = f\|_{0,1}$	$T_9^{x_i,x_j}(f) \iff f\|_{0,0} = \neg f\|_{0,1}$
$T_4^{x_i,x_j}(f) \iff f\|_{1,0} = f\|_{1,1}$	$T_{10}^{x_i,x_j}(f) \iff f\|_{1,0} = \neg f\|_{1,1}$
$T_5^{x_i,x_j}(f) \iff f\|_{0,0} = f\|_{1,0}$	$T_{11}^{x_i,x_j}(f) \iff f\|_{0,0} = \neg f\|_{1,0}$
$T_6^{x_i,x_j}(f) \iff f\|_{0,1} = f\|_{1,1}$	$T_{12}^{x_i,x_j}(f) \iff f\|_{0,1} = \neg f\|_{1,1}$

We previously presented an anytime algorithm for symmetry detection for Boolean functions represented as ROBDDs [16]. The algorithm sought to address some of the drawbacks associated with existing methods that have been proposed for ROBDDs. One problem that we have found is that the running time of these algorithms [12], [17] can exceed 12 hours on some ROBDDs of less than a million nodes. Variable reordering can reduce the size of an ROBDD and thereby reduce the cost of symmetry detection. However, it is imprudent to rely on variable reordering alone to make symmetry detection tractable since variable reordering techniques can themselves be prohibitively expensive and of course, even after reordering, there is no guarantee that the size of the ROBDD will actually be smaller. In fact even improving the variable ordering is NP-complete [18], and is also inapproximable within a constant factor [19] (that is, if for every given $\epsilon > 0$, there exists a polynomial-time algorithm for reordering variables so as to obtain an ROBDD whose size is not larger than $1 + \epsilon$ times that of the minimal size, then it follows that $P = NP$). From the perspective of algorithm design, there are at least two ways forward: develop a faster symmetry detection algorithm; recast symmetry detection so that it can be solved with an anytime algorithm. Anytime algorithms arise in engineering tasks when it is more attractive to find an acceptable answer in a reasonable amount of time rather than the optimal answer in an exorbitant amount of time. In the context of symmetry detection the challenge is therefore to devise an efficient algorithm that incrementally detects pairs of symmetric variables until some given time bound is exceeded.

Thus far, the only incremental algorithms that have been proposed for symmetry detection in ROBDDs are those based on naïve co-factor computation [9], [20], but alas, this approach is inefficient. The algorithm of Panda *et al.* [8] can be considered to be incremental and does not require co-factor computation. Instead, the algorithm is formulated in terms of dynamic variable reordering [21]. This approach is incomplete for the purposes of symmetry detection, since the algorithm may not detect all symmetric variable pairs if variable reordering is prematurely terminated. The most efficient algorithms proposed thus far for symmetry detection [12], [17] are monolithic in that they provide no opportunity for early termination, and yet can sometimes require significant runtime. In this paper we present a class of efficient anytime algorithms for classical and generalized symmetry detection. For clarity, we summarize our contributions as follows:

- The paper presents an incremental, anytime algorithm for first-order classical symmetry detection. Even considering the complexity of all the underlying set operations, the algorithm is in $O(n^3+n|G|+|G|^3)$ where n is the number of variables and $|G|$ the number of nodes in the ROBDD.
- The paper explains how an incremental anytime approach offers special opportunities for optimization, in that classical assymetry/symmetry sieves can precede the algorithm and assymetry/symmetry propagation techniques can be inserted into the main loop of the algorithm.
- The paper proposes a computationally lightweight technique that often improves the proportion of symmetries found early on in the operation of the algorithm.
- The paper shows how to refine the anytime algorithm so as to detect generalized symmetries. An algorithm for simultaneously detecting all $T_1, \ldots, T_{12}$-symmetries is presented which resides in $O(n^3 + n^2|G| + |G|^3)$. This algorithm is underpinned by new symmetry relationships which take the form, that if $T_p^{x_i,x_j}(f)$ and $T_q^{x_j,x_k}(f)$ hold then $T_r^{x_i,x_j}(f)$ holds where T_p, T_q and T_r denote one of the 12 generalized symmetry types. Only a few of these transitivity results have been previously reported [22] and these results could well find application in other symmetry detection problems [23].
- The paper shows that symmetry detection does not require the creation of intermediate ROBDDs and that anytime generality need not compromise efficiency.

The remainder of this paper is structured thusly: Section II presents the necessary preliminaries and Section III surveys the related work. Section IV presents an anytime symmetry detection algorithm for classical symmetries. Section V explains how the multi-pass nature of the algorithm can be exploited with asymmetry/symmetry propagation. Section VI extends the anytime approach to the detection of generalized symmetries. Section VII quantifies the cost of anytime symmetry detection and Section VIII concludes.

II. Preliminaries

In this paper we consider completely specified Boolean functions $f : \mathit{Bool}^n \rightarrow \mathit{Bool}$ where $\mathit{Bool} = \{0,1\}$ that are conventionally written as Boolean formulae defined over a variable set $X = \{x_1, \ldots, x_n\}$. The satisfy-count of an n-ary Boolean function f is defined as $\|f\| = |\{(b_1, \ldots, b_n) \mid f(b_1, \ldots, b_n) = 1\}|$ [24]. The (Shannon) co-factor of a function f w.r.t a variable x_i and a Boolean constant $b \in \mathit{Bool}$ is defined by $f|_{x_i \leftarrow b} = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$. Multiple variable co-factors, denoted $f|_{x_{i_1} \leftarrow b_1, \ldots, x_{i_m} \leftarrow b_m}$, can be defined inductively as $f_0 = f$, $f_j = f_{j-1}|_{x_{i_j} \leftarrow b_j}$ and $f|_{x_1 \leftarrow b_1, \ldots, x_m \leftarrow b_m} = f_m$.

A BDD is a rooted directed acyclic graph where each internal node is labeled with a Boolean variable x_i. Each internal node has one successor node connected via an edge labeled 0, and another successor connected via an edge labeled 1. Each leaf node is either the Boolean constant 0 or 1. The Boolean function represented by a BDD can be evaluated for a given variable assignment $\{x_1 \rightarrow b_1, \ldots, x_n \rightarrow b_n\}$ where $b_i \in \mathit{Bool}$ by traversing the graph from the root, taking the 1 edge at a node when the variable x_i is assigned to 1 and the 0 edge when the variable x_i is assigned to 0. The leaf reached in this traversal indicates the value of the Boolean function for the assignment. An OBDD is a BDD with the restriction that the label of an internal node, x_i, is always less than the label of any internal node reachable via its successors, x_j, that is, $i < j$. An ROBDD is an OBDD with the additional constraint that the two successor nodes of any internal node represent different Boolean functions, and that distinct internal nodes also represent distinct Boolean functions. Note that any internal node of an ROBDD is itself the root of an ROBDD.

Each of the 12 predicates $T_i^{x_j,x_k}(f)$ of Table 1 asserts a symmetry property of a Boolean function f where the predicate $T_i^{x_j,x_k}(f)$ is interpreted as stating that the Boolean function f is T_i-symmetric in the variable pair (x_j, x_k). Strictly, an ROBDD g is not a Boolean function but rather a representation of one. Therefore to assert symmetry properties of the function f that underlies a given ROBDD g, we define $T_i^{x_j,x_k}(g)$ to hold whenever $T_i^{x_j,x_k}(f)$ holds. Moreover, we shall say that a ROBDD g is T_i-symmetric in the variable pair (x_j, x_k) iff $T_i^{x_j,x_k}(g)$ holds, and dually g is T_i-asymmetric in the variable pair (x_j, x_k) iff $T_i^{x_j,x_k}(g)$ does not hold.

III. Related Work

Early work on detecting symmetric variables in Boolean functions has focussed on the computation of co-factor pairs, that is all $n^2 - n$ possible co-factors, where n is the number of variables. Symmetry is detected by checking their equivalence [20]. The use of ROBDDs to represent Boolean functions enables not only the efficient computation of co-factors, but also equivalence to be checked in constant time. However, repeated co-factoring involves the creation and deletion of many intermediate ROBDD nodes and for very large ROBDDs this overhead can be prohibitive. This method is often referred to as the naïve method [20]. Möller, Mohnke and Weber [20] thus advocate the use of preprocessing algorithms — sieves — that detect pairs of asymmetric variables. These linear-time sieves significantly reduce the number of co-factor pairs that need to be computed. In general, however, methods built upon such sieves still require naïve co-factor computation, that is, calls to the standard co-factoring algorithm [24] the complexity of which is in $O(|G| \lg |G|)$.

Because of the cost of repeated co-factoring, many symmetry detection methods endeavor to avoid naïve co-factor computation. Möller *et al.* [20] and Panda *el al.* [8] detect all symmetries between variables adjacent in the variable order with an algorithm in $O(|G|)$. Panda *et al.* [8] modify Rudell's dynamic variable reordering algorithm [21] to detect symmetries between variables that become adjacent when one of the variables is repositioned in the ROBDD variable ordering. Symmetric variables are then grouped, and any subsequent reordering that is applied is required to preserve a contiguous variable ordering within each group. This approach to symmetry detection does not require naïve co-factor computation, but there is no guarantee that all symmetries will be found if variable reordering is prematurely terminated.

The algorithm of Mishchenko [17] can detect all symmetric variable pairs in a ROBDD with just $O(|G|^3)$ set operations. Zero suppressed binary decision diagrams (ZBDDs) [25] are used to compactly represent a collection of sets of symmetric variable pairs. However, since each set can potentially contain $O(n^2)$ elements one would expect Mishchenko's algorithm to at least reside in $O(n^2|G|^3)$ and possibly even a higher complexity class when all set operations are considered.

The generalization of symmetries is a recent development and has received much attention [12], [13], [15], [26]. This move to generalized symmetries has inevitably brought with it the requirement for efficient algorithms to compute them [12], [26]. It is straightforward to extend the naïve approach of symmetry detection to all generalized symmetries in Table I with only a worst-case twofold increase in the amount of work required. This is because classical symmetry detection requires calculating the co-factors $f|_{x_i \leftarrow 1, x_j \leftarrow 0}$ and $f|_{x_i \leftarrow 0, x_j \leftarrow 1}$ whereas generalized symmetries over two variables only require the co-factors $f|_{x_i \leftarrow 0, x_j \leftarrow 0}$ and $f|_{x_i \leftarrow 1, x_j \leftarrow 1}$ to be additionally computed. (The amount of work required to compute an equivalence check, such as $f|_{x_i \leftarrow 0, x_j \leftarrow 0} = f|_{x_i \leftarrow 1, x_j \leftarrow 1}$, is negligible and a check that involves negation, such as $f|_{x_i \leftarrow 0, x_j \leftarrow 0} = \neg f|_{x_i \leftarrow 1, x_j \leftarrow 1}$, is also in $O(1)$ for ROBDDs with complement edges [27].) This twofold increase in work is disproportionate to the twelvefold increase in the number of symmetries that can be detected, however, the overhead of repeated co-factoring is still prohibitive. Consequently, symmetry detection methods for generalized symmetries have progressed along the same lines as those for classical symmetries: the algorithm of Zhang *et al.* [12] mirrors the design of Mishchenko [17], but is altered to perform multiple passes for each of the different symmetry types. Hence, the algorithm of Zhang *et al.* has the same worst-case complexity of that of Mishchenko, disregarding constant factors.

An interesting thread of related research focusses on the problem of extracting symmetries from Boolean functions that are not represented as ROBDDs [23], [28].

IV. Anytime Symmetry Detection Algorithm

In this section we describe our anytime approach to classical symmetry detection. For pedagogical purposes we first present Algorithm 1 which is our simplest algorithm for anytime symmetry detection. In the section that follows, we build on Algorithm 1 by incorporating optimizations that exploit its anytime nature. Algorithm 1 takes as input an ROBDD f and returns a set of index pairs $S = \{(i,j) \mid T_1^{x_i,x_j}(f)\}$ that represent the set of T_1-symmetric variable pairs. The algorithm is composed of two separate procedures: `FindAsymmetry` and `RemoveAsymmetry`. `FindAsymmetry`(f) performs two depth-first traversals over the ROBDD f to detect pairs of variables (x_i, x_j) that are provably asymmetric with respect to T_1. `RemoveAsymmetry`(f, i, C) filters a set of variable indices C whose symmetry relationship with variable x_i is unknown to return the set $C' \subseteq C$ that represents those variables x_j that are T_1-symmetric with x_i.

Algorithm 1 SymmetricPairs(f)

$A \leftarrow$ `FindAsymmetry`(f)
$S \leftarrow \emptyset$
for $i = 1$ **to** $n-1$ **do**
  $C \leftarrow \{ j \mid (i,j) \notin (A \cup S) \wedge i < j\}$
  $D \leftarrow$ `RemoveAsymmetry`(f, i, C)
  $A \leftarrow A \cup \{(i,l),(l,i) \mid l \in C \setminus D\}$
  $S \leftarrow S \cup \{(i,l),(l,i) \mid l \in D\}$
return S

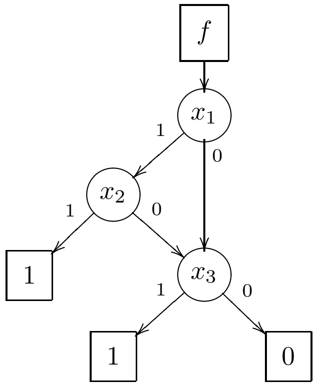
The call to `FindAsymmetry` initializes the set of asymmetric variable pairs A such that $A \subseteq \{(i,j) \mid \neg T_1^{x_i,x_j}(f)\}$. The set C is constructed of indices for those variables whose T_1-symmetry relation with x_i is as yet undetermined. The set of T_1-symmetric variables D returned from `RemoveAsymmetry` and its complement $C \setminus D$ are used to extend S and A respectively. The main loop only requires $n-1$ iterations because $C = \emptyset$ when $i = n$. The algorithm that initializes A is justified by lemmata that detail how T_1-symmetric variables place structural constraints on ROBDDs [9][lemmata 5 and 6]. We state these lemmata below for completeness:

Lemma 1. *If an ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_1-symmetric in the pair (x_i, x_j) and $i < j$, then every ROBDD rooted at a node labeled x_i must contain a node labeled x_j.*

Lemma 2. *If an ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_1-symmetric in the pair (x_i, x_j) and $i < j$, then every path from the root of f to a node labeled x_j must visit a node labeled x_i.*

Lemmata 1 and 2 provide two conditions under which asymmetry can be observed. For any given node labeled x_i we can compute the set of all variables x_j that appear in a ROBDD that is rooted at that node, and any variable not appearing in this set is necessarily T_1-asymmetric with x_i. Furthermore, for any given node labeled x_j, we can compute the set of all variables x_i that appear on *all* paths from the root of the ROBDD to the node, and any variable not appearing in this set is T_1-asymmetric with x_j. These asymmetry conditions can be checked together in just two depth-first traversals of the ROBDD, each traversal taking $O(n|G|)$ time since each node is visited singly and at most n variables need be considered.

The symmetry relations between the variables are computed in a series of passes. The validity of this decomposition is justified by the proposition:



1: The ROBDD f for the formula $(x_1 \wedge x_2) \vee x_3$

Proposition 1. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_1-symmetric in the pair (x_i, x_j) and $i < j$ iff*

1) *every ROBDD rooted at a node labeled x_i is T_1-symmetric in (x_i, x_j) and,*
2) *every path from the root of f to a node labeled x_j passes through a node labeled x_i.*

Proof.

- Consider the *if* direction.
  - Since f is T_1-symmetric in the variable pair (x_i, x_j), $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$ for all $\mathbf{b}_1 \in \mathit{Bool}^{i-1}$, $\mathbf{b}_2 \in \mathit{Bool}^{j-i-1}$ and $\mathbf{b}_3 \in \mathit{Bool}^{n-j}$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ hence $g|_{x_i \leftarrow 1, x_j \leftarrow 0} = g|_{x_i \leftarrow 0, x_j \leftarrow 1}$.
  - Suppose for the sake of a contradiction that there exists a path from the root to a node labeled x_j that does not pass through a node labeled x_i. Thus, let $g = f(\mathbf{b}_1, 0, \mathbf{b}_2, x_j, \ldots, x_n) = f(\mathbf{b}_1, 1, \mathbf{b}_2, x_j, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathit{Bool}^{i-1}$ and $\mathbf{b}_2 \in \mathit{Bool}^{j-i-1}$. Thus $g|_{x_j \leftarrow 0}(\mathbf{b}_3) = g|_{x_j \leftarrow 1}(\mathbf{b}_3)$ for all $\mathbf{b}_3 \in \mathit{Bool}^{n-j}$. Hence $g|_{x_j \leftarrow 0} = g|_{x_j \leftarrow 1}$ which is a contradiction since g is reduced.
- Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathit{Bool}^{i-1}$, $\mathbf{b}_2 \in \mathit{Bool}^{j-i-1}$ and $\mathbf{b}_3 \in \mathit{Bool}^{n-j}$ such that $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.
  - Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$. Thus g is labeled x_i, hence there exists some $\mathbf{b}_2$ and $\mathbf{b}_3$ such that $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $g|_{x_i \leftarrow 0}(\mathbf{b}_2, 1, \mathbf{b}_3) = 0$ as required.
  - Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$. Hence g is not labeled x_i. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) \neq h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ as required.

  The $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$ case follows analogously. □

One may wonder if the second condition in the proposition is actually necessary. Figure 1 illustrates that this condition cannot be relaxed. Observe that the variable pair (x_2, x_3) is T_1-symmetric in the ROBDD rooted at x_2, moreover the pair (x_2, x_3) is T_1-symmetric for *every* ROBDD rooted at a node labeled x_2. However, the pair (x_2, x_3) is T_1-asymmetric in the ROBDD f, and indeed there exists a path from the root of f to the node x_3 that does not visit a node labeled x_2.

The proposition allows exhaustive checking to be decomposed into a series of passes; one pass for each variable x_i. Observe that when the loop is entered in Algorithm 1, `FindAsymmetry` has already added all the pairs (i, j) to A such that there exists a path from the root to a node labeled x_j which does not pass through a node labeled x_i. An index j for such a pair cannot arise in C. Hence it remains to remove those indices $j \in C$ which violate the first condition of the proposition, that is, those $j \in C$ for which f is T_1-asymmetric in the pair (x_i, x_j). This is precisely the role of `RemoveAsymmetry`(f, i, C) in Algorithm 2 where the parameter i delineates the variable under consideration in the pass. The algorithm uses the function `index`(f) which merely returns the index of the root of an ROBDD f, that is, i if the root of f is labeled x_i.

Algorithm 2 RemoveAsymmetry(f, i, C)

if $C = \emptyset \vee f = \mathbf{true} \vee f = \mathbf{false}$ **then**
  return C
$j \leftarrow$ `index`(f)
if $j > i$ **then**
  return C
else if $j = i$ **then**
  return `RemoveAsymmetryVar`$(f|_{x_i \leftarrow 0}, f|_{x_i \leftarrow 1}, C)$
else
  $C \leftarrow$ `RemoveAsymmetry`$(f|_{x_j \leftarrow 0}, i, C)$
  return `RemoveAsymmetry`$(f|_{x_j \leftarrow 1}, i, C)$

An index j should be removed from C whenever $f|_{x_i \leftarrow 0, x_j \leftarrow 1} \neq f|_{x_i \leftarrow 1, x_j \leftarrow 0}$. This T_1-asymmetry check is satisfied if there exists $\mathbf{a} \in \mathit{Bool}^{i-1}$ and $\mathbf{b} \in \mathit{Bool}^{j-i-1}$ such that $f(\mathbf{a}, 0, \mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq f(\mathbf{a}, 1, \mathbf{b}, 0, x_{j+1}, \ldots, x_n)$ where i refers to the position between $\mathbf{a}$ and $\mathbf{b}$. If $k =$ `index`(f), $f_0 = f|_{x_k \leftarrow 0}$ and $f_1 = f|_{x_k \leftarrow 1}$ then showing $f(\mathbf{a}, 0, \mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq f(\mathbf{a}, 1, \mathbf{b}, 0, x_{j+1}, \ldots, x_n)$ amounts to detecting either $f_0(\mathbf{a}, 0, \mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq f_0(\mathbf{a}, 1, \mathbf{b}, 0, x_{j+1}, \ldots, x_n)$ or $f_1(\mathbf{a}, 0, \mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq f_1(\mathbf{a}, 1, \mathbf{b}, 0, x_{j+1}, \ldots, x_n)$ for some (smaller) $\mathbf{a} \in \mathit{Bool}^{i-2}$. This recursive reduction explains the recursive nature of `RemoveAsymmetry`. The test $j > i$ implements a form of early termination since if $j > i$ there is no opportunity for removing any index from C. The leaves $\mathbf{true}$ and $\mathbf{false}$ also trigger early termination.

At the heart of `RemoveAsymmetry` is a call to `RemoveAsymmetryVar`$(f|_{x_i \leftarrow 0}, f|_{x_i \leftarrow 1}, C)$ which is applied to an ROBDD whose root is labeled with the variable x_i. When a call to `RemoveAsymmetryVar` is initially encountered, its first and second parameters are $g_0 = g|_{x_i \leftarrow 0}$ and $g_1 = g|_{x_i \leftarrow 1}$. At this point, it remains to search for some $\mathbf{b} \in \mathit{Bool}^{j-i-1}$ such that $g_0(\mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq g_1(\mathbf{b}, 0, x_{j+1}, \ldots, x_n)$. This is in turn realized by showing either $g_{00}(\mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq g_{10}(\mathbf{b}, 0, x_{j+1}, \ldots, x_n)$ or $g_{01}(\mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq g_{11}(\mathbf{b}, 0, x_{j+1}, \ldots, x_n)$ for some (smaller) $\mathbf{b} \in \mathit{Bool}^{j-i-2}$ where $g_{00} = g_0|_{x_{i+1} \leftarrow 0}$, $g_{10} = g_1|_{x_{i+1} \leftarrow 0}$, $g_{01} = g_0|_{x_{i+1} \leftarrow 1}$ and $g_{11} = g_1|_{x_{i+1} \leftarrow 1}$. A

Algorithm 3 RemoveAsymmetryVar(g_0, g_1, C)

```
if g_0 = true ∨ g_0 = false then
    j ← ∞
else
    j ← index(g_0)
if g_1 = true ∨ g_1 = false then
    r ← ∞
else
    r ← index(g_1)
if C = ∅ ∨ j = r = ∞ then
    return C
else if j = r then
    (l, g_00, g_01, g_10, g_11) ← (j, g_0|x_j←0, g_0|x_j←1, g_1|x_r←0, g_1|x_r←1)
else if j < r then
    (l, g_00, g_01, g_10, g_11) ← (j, g_0|x_j←0, g_0|x_j←1, g_1, g_1)
else
    (l, g_00, g_01, g_10, g_11) ← (r, g_0, g_0, g_1|x_r←0, g_1|x_r←1)
if g_01 ≠ g_10 then
    C ← C \ {l}
C ← RemoveAsymmetryVar(g_00, g_10, C)
return RemoveAsymmetryVar(g_01, g_11, C)
```

recursive formulation of RemoveAsymmetryVar can be obtained from this recursive reduction. When both g_0 and g_1 are leaf nodes, no further reduction can be applied and RemoveAsymmetryVar terminates.

The three cases in Algorithm 3 are required to accommodate the reduction inherent in ROBDDs. The $j = r$ condition selects the case when g_0 and g_1 are labeled with the same variable x_j. In this case we compute $g_0|_{x_j \leftarrow 1}$ and $g_1|_{x_j \leftarrow 0}$ and check that $g_0|_{x_j \leftarrow 1} \neq g_1|_{x_j \leftarrow 0}$. If the check is satisfied j is removed from C. When $j < r$ the co-factor $g_1|_{x_j \leftarrow 1} = g_1$ hence the asymmetry check $g_0|_{x_j \leftarrow 1} \neq g_1|_{x_j \leftarrow 0}$ reduces to $g_0|_{x_j \leftarrow 1} \neq g_1$. If this check is satisfied j is removed from C. The $r < j$ case is analogous except that r is removed.

Caching can be applied to ensure that the function RemoveAsymmetryVar is not called twice on the same pair of ROBDDs g_0 and g_1. Moreover, the complexity of a call to RemoveAsymmetryVar is in $O(|G|^2)$ if C is represented as an array of n Booleans. Then computing $C \setminus \{l\}$ is in $O(1)$, as is the test $C = \emptyset$ when C is augmented with a counter to record $|C|$. Overall, RemoveAsymmetryVar can only be invoked a total of $|G|$ times from within Algorithm 1, thus RemoveAsymmetryVar contributes $O(|G|^3)$ to the overall running time. The $n-1$ calls to RemoveAsymmetry cumulatively cost $O(n|G|)$. Returning to the main loop of Algorithm 1, observe that the sets A and S can be augmented in $O(n)$ time when D is also represented as an array of n Booleans and A and S are represented as $n \times n$ adjacency matrices. Algorithm 1 is therefore in $O(n^2 + n|G| + |G|^3)$. Interestingly, although this improves on the algorithm of Mishchenko when set operations are considered, it does not improve on the naïve co-factor computation method [9], [20] which resides in $O(n^2|G| \lg(|G|))$.

V. Optimized Anytime Symmetry Detection

In this section we propose a series of optimizations for Algorithm 1. The resulting refined algorithm retains the incremental nature of the original algorithm, and shows how incrementality can be exploited by several optimizations. These optimizations seek to reduce the size of the set C, and hence the running time of the call RemoveAsymmetry(f, i, C), by enriching the sets A and S on-the-fly before, and between, iterations of the main loop. The symmetry sieve algorithms proposed by [9], [10], [20] suggest a way to refine the sets A and S before the loop is entered. Furthermore, it is possible to take advantage of the transitivity of the T_1-symmetry relation to add further pairs to A and S between iterations. The novelty is not in the optimizations themselves, but rather that an anytime reformation of symmetry detection naturally accommodates various useful optimizations [9], [10], [20]. The optimized algorithm listed in Algorithm 4 takes an ROBDD f and returns the set S of T_1-symmetric variable pairs.

Algorithm 4 OptimizedSymmetricPairs(f)

```
A' ← FindAsymmetry(f)
M ← SatisfyCounts(f)
for i = 1 to n do
    for j = i + 1 to n do
        if M(i) ≠ M(j) then
            A' ← A' ∪ {(i, j), (j, i)}
(A, S) ← FindAdjSymmetry(f)
(A, S) ← (A ∪ A', S \ A')
for i = 1 to n − 2 do
    (A, S) ← SymmetryClosure(A, S)
    C ← { j | (i, j) ∉ (A ∪ S) ∧ i + 1 < j}
    D ← RemoveAsymmetry(f, i, C)
    A ← A ∪ {(i, l), (l, i) | l ∈ C \ D}
    S ← S ∪ {(i, l), (l, i) | l ∈ D}
return S
```

SatisfyCounts(f) returns a mapping M from variable indices to a natural number that can be used to distinguish pairs of T_1-asymmetric variables, that is, if $M(i) \neq M(j)$ then (x_i, x_j) are T_1-asymmetric. FindAdjSymmetry(f) returns two sets of index pairs A and S where $\{(i,j) \mid \neg T_1^{x_i,x_j}(f) \wedge j = i+1\} \subseteq A \subseteq \{(i,j) \mid \neg T_1^{x_i,x_j}(f)\}$ and $S = \{(i,j) \mid T_1^{x_i,x_j}(f) \wedge j = i+1\}$. Since the procedure FindAdjSymmetry finds all adjacent T_1-symmetric and T_1-asymmetric pairs, the number of loop iterations can be relaxed from $n-1$ to $n-2$. SymmetryClosure(A_1, S_1) takes as input two sets A_1 and S_1 of variable pairs known to be T_1-asymmetric and T_1-symmetric respectively. Then, by reasoning about transitivity, a pair of sets (A_2, S_2) is computed which are T_1-symmetric and T_1-asymmetric such that $A_2 \supseteq A_1$ and $S_2 \supseteq S_1$. The procedures SatisfyCounts, FindAdjSymmetry and SymmetryClosure are detailed in Sections V-A, V-B and V-C respectively. Section V-D presents some heuristics which endeavor to increase the proportion of T_1-symmetric variable pairs that are discovered early on in the execution of the main loop of Algorithm 4.

A. Satisfy Counts

A consequence of T_1-symmetry, which can also be used to detect T_1-asymmetry [10], relates the satisfy count of one positive co-factor of a variable to the satisfy count of another:

Lemma 3. *If a Boolean function f over a set of variables $\{x_1, \ldots, x_n\}$ is T_1-symmetric in the pair (x_i, x_j), then $\|f|_{x_i \leftarrow 1}\| = \|f|_{x_j \leftarrow 1}\|$.*

Computing the satisfy counts of all co-factors can be realized using a single depth-first traversal of the ROBDD in $O(n|G|)$ time [10]. Finding the resultant asymmetries additionally requires n^2 comparisons in Algorithm 4, and thus the overall complexity of this sieve is $O(n^2 + n|G|)$.

B. Adjacent Symmetries

The following result follows immediately from Proposition 1 and details a special case of symmetry which relates to variables that are adjacent in the ROBDD ordering:

Corollary 1. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_1-symmetric in the pair (x_i, x_{i+1}) iff*

1) *every ROBDD rooted at a node labeled x_i is T_1-symmetric in (x_i, x_{i+1}) and,*
2) *every path from the root of f to a node labeled x_{i+1} passes through a node labeled x_i.*

The force of this result is that the equivalence $f|_{x_i \leftarrow 0, x_{i+1} \leftarrow 1} = f|_{x_i \leftarrow 1, x_{i+1} \leftarrow 0}$ can be checked in $O(|G|)$ time for all adjacent variable pairs [20]. In fact Proposition 1 leads to a further result that can detect T_1-asymmetric variable pairs that are not necessarily adjacent in the variable ordering:

Corollary 2. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_1-asymmetric in the pair (x_i, x_k) if there exists a node g in f labeled x_i with successor nodes labeled x_k and x_l where $i+1 < k \leq l$ and $g|_{x_i \leftarrow 0, x_k \leftarrow 1} \neq g|_{x_i \leftarrow 1, x_k \leftarrow 0}$.*

These non-consecutive T_1-asymmetric pairs can be detected in $O(|G|)$ time. Of course, the first $O(|G|)$ tactic for enriching A and S can only be deployed in conjunction with `FindAsymmetry`; the second tactic is independent of `FindAsymmetry`.

C. Symmetry Closure

The following lemma can be obtained by recalling that a function f remains unchanged under the switching of any pair of T_1-symmetric variables:

Lemma 4. *If a Boolean function f over a set of variables $X = \{x_1, \ldots, x_n\}$ is T_1-symmetric in the pairs (x_i, x_j) and (x_j, x_k) then f is also T_1-symmetric in the pair (x_i, x_k).*

This transitivity result provides a way of enriching the set S, that is, if $(x_i, x_j), (x_j, x_k) \in S$ then it follows that (x_i, x_k) is also a T_1-symmetric pair, hence S can be enriched with (x_i, x_k). Further, if $(x_i, x_j) \in S, (x_i, x_k) \in A$ then it follows that the pair (x_j, x_k) is T_1-asymmetric, that is, A can be enriched with (x_j, x_k). This follows since if (x_j, x_k) is T_1-symmetric then by the lemma it follows that (x_i, x_k) is T_1-symmetric, which is a contradiction. Adding those variable pairs to A and S which can be inferred through transitivity is not dissimilar to computing the transitive closure of a binary relation. This motivates adapting an algorithm such as the Floyd-Warshall all-pairs-shortest-path algorithm [29], [30] to this task. The complexity of this transitive algorithm is in $O(n^3)$ when A and S are represented as $n \times n$ adjacency matrices. Each iteration of the main loop of Algorithm 4 incurs an additional call to `SymmetryClosure`, which computes the transitive closure, and pushes the overall complexity into $O(n^4 + n|G| + |G|^3)$. Recall that `SatisfyCounts` and `FindAdjSymmetry` are in $O(n|G|)$ and $O(|G|)$ respectively which have no impact on the overall asymptotic complexity. However, although the Floyd-Warshall is attractive because of its simplicity, the complexity can be reduced to $O(n^3 + n|G| + |G|^3)$, or even lower, by substituting Floyd-Warshall with an incremental (online) transitive closure algorithm [31].

D. Variable Choice Heuristics

The astute reader may have noticed that the correctness of Algorithm 4 is not compromised by the order in which variables are considered in the main loop. One may wonder therefore if considering variables in a different order can speed up the algorithm. One natural approach is to choose a variable x_i that maximizes $|\{(x_i, x_j) \notin (A \cup S) \wedge i < j\}|$. The rationale behind this greedy heuristic is to ensure that the call to `RemoveAsymmetry` resolves the maximal number of variable pairs whose T_1-symmetry relation is unknown. The dual of this heuristic is to choose a variable x_i for which unknowns remain which minimizes $|\{(x_i, x_j) \notin (A \cup S) \wedge i < j\}|$. Motivation for this heuristic comes from literature [32] on computing signatures for Boolean functions so as to determine input correspondence. This is the problem of determining whether the variables of one ROBDD can be reordered so that the resulting ROBDD is equivalent to another. It has been observed that if the currently known asymmetry sieves [10], [20] leave only a handful of pairs for which a symmetry is unknown, then these variables are likely to be involved in some symmetry relationship [32]. Therefore, focusing `RemoveAsymmetry` on the variable with the least unknowns is likely to discover T_1-symmetries. We call these two heuristics $\max$ and $\min$ respectively. It should be pointed out that for both these heuristics, a variable can be chosen in $O(n)$ time by maintaining a counter for each variable x_i that records the number of unknowns, that is, $|\{(x_i, x_j) \notin (A \cup S) \wedge i < j\}|$. The counter for x_i is decremented each time a pair (x_i, x_j) is added to A or S. The cumulative overhead of running the heuristic over the loop body is in $O(n^2)$ which is absorbed into the asymptotic running time of the algorithm.

VI. Generalized Anytime Symmetry Detection

In this section we show how to extend the anytime algorithm presented in the previous section to also detect the generalized symmetry types given in Table I. The section presents a series of novel results which detail the structural constraints that generalized symmetries place on an ROBDD. The force of these results is that they justify the construction of asymmetry sieves since an ROBDD cannot possess a symmetry if the structural constraints that follow from that symmetry do not hold. These results also explain how generalized symmetry detection can be decomposed into a series of passes. In addition, the section presents a number of novel transitivity results of the form, that if $T_p^{x_i,x_j}(f)$ and $T_q^{x_j,x_k}(f)$ hold then $T_r^{x_i,x_j}(f)$ holds where T_p, T_q and T_r denote one of the 12 generalized symmetry types. These transitivity results allow assymetry/symmetry propagation to be inserted between the passes of any anytime generalized symmetry detection algorithm.

Algorithm 5 takes as input an ROBDD f and returns the set of triples $S = \{(i, j, k) \mid T_k^{x_i,x_j}(f)\}$. The algorithm is composed of three distinct procedures. `FindFastSymmetry`(f) returns a pair (A, S) such that $A = \{(i, j, k) \mid \neg T_k^{x_i,x_j}(f) \wedge k \in K\}$ and $S = \{(i, j, k) \mid T_k^{x_i,x_j}(f) \wedge k \in K\}$ where $K = \{3, 4, 9, 10\}$. `FindSlowAsymmetry`(f) returns a set $A' \subseteq \{(i, j, k) \mid \neg T_k^{x_i,x_j}(f) \wedge k \in K'\}$ where $K' = \{1, \ldots, 12\} \setminus K$. In an analogous fashion to before, `GeneralRemoveAsymmetry`(f, i, C) filters a set of pairs C to return a subset $C' \subseteq C$. If the T_k-symmetry relationship between the variables x_i and x_j is presently unknown then $(j, k) \in C$. The returned set $C' \subseteq C$ is precisely those pairs $C' = \{(j, k) \in C \mid T_k^{x_i,x_j}(f) \wedge k \in K'\}$.

Algorithm 5 GeneralizedSymmetricPairs(f)

$(A, S) \leftarrow$ `FindFastSymmetry`(f)
$A \leftarrow A \cup$ `FindSlowAsymmetry`(f)
for $i = 1$ **to** $n - 1$ **do**
  $C \leftarrow \{\, (j, k) \mid (i, j, k) \notin (A \cup S) \wedge i < j\}$
  $D \leftarrow$ `GeneralRemoveAsymmetry`(f, i, C)
  $A \leftarrow A \cup \{(i, l, k),\ (l, i, k) \mid (l, k) \in C \setminus D\}$
  $S \leftarrow S \cup \{(i, l, k), (l, i, k) \mid (l, k) \in D\}$
return S

A. Fast Symmetries

Interestingly, some types of generalized symmetry are easier to compute than others. In fact, T_3 and T_4-symmetries and T_9 and T_{10}-symmetries can be computed in $O(n|G|)$ and $O(n^2|G|)$ respectively, utilizing the following two propositions. The proofs for the results reported in this section are similar in spirit to that of Proposition 1 and therefore, for reasons of continuity, are relegated to an accompanying technical report [33].

Proposition 2. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_3-symmetric (resp. T_4-symmetric) in the pair (x_i, x_j) and $i < j$ iff*

1) *if whenever an ROBDD g occurs in f at a node labeled x_i then $g|_{x_i \leftarrow 0}$ (resp. $g|_{x_i \leftarrow 1}$) does not contain a node labeled x_j and,*
2) *every path from the root of f to a node labeled x_j passes through a node labeled x_i.*

Proposition 3. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_9-symmetric (resp. T_{10}-symmetric) in the pair (x_i, x_j) and $i < j$ iff*

1) *if whenever an ROBDD g occurs in f at a node labeled x_i then every path through $g|_{x_i \leftarrow 0}$ (resp. $g|_{x_i \leftarrow 1}$) visits a node h labeled x_j such that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$ and,*
2) *every path from the root of f to a node h labeled x_j which does not visit a node labeled x_i, satisfies the property that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$.*

The first and second conditions of Proposition 2 can be checked in two depth-first traversals both requiring $O(n|G|)$ time and thus all T_3 and T_4-symmetries can be detected in $O(n|G|)$ time overall. Detecting T_9 and T_{10}-symmetries resides in $O(n^2|G|)$ since Proposition 3 implies that T_9 and T_{10}-asymmetries can be found by systematically searching through all pairs of variables (x_i, x_j), checking that f includes a path that neither contains x_i nor x_j. These propositions assert that T_3, T_4, T_9 and T_{10}-symmetries are surprisingly tractable, and therefore suggest that these symmetries are particularly interesting for those applications where it is not necessary to compute all types of generalized symmetry [10]–[12].

B. Slow Symmetries

Computing the remaining generalized symmetries, namely $T_2, T_5, T_6, T_7, T_8, T_{11}$ and T_{12}, requires more effort. The following four propositions explain how each of these symmetry relations can be computed in a series of passes where each pass computes all the symmetry types for each variable x_i.

Proposition 4. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_2-symmetric in the pair (x_i, x_j) and $i < j$ iff*

1) *every ROBDD rooted at a node labeled x_i is T_2-symmetric in (x_i, x_j) and,*
2) *every path from the root of f to a node labeled x_j passes through a node labeled x_i.*

Like before, the proposition asserts that all T_2-symmetries can be found in two stages. The first stage, a lightweight preprocessing step, marks a pair (x_i, x_j) as T_2-asymmetric if f contains a path to a node labeled x_j that does not pass through a node labeled x_i. The second stage, which amounts to exhaustive search, examines each node labeled x_i and checks whether the ROBDD rooted at that node is T_2-asymmetric in (x_i, x_j). The first check is one of a number carried out by the call to `GeneralRemoveAsymmetry` in the main loop of Algorithm 5. The second check is realized in the function `FindSlowAsymmetry` which precedes the main loop. Thus, paradoxically, the first check is applied chronologically after the second check. `GeneralRemoveAsymmetry` and `FindSlowAsymmetry` also carry out checks to verify the first and second conditions of both Propositions 6 and 7. The simple structure of Proposition 5 permits T_5 and T_6 symmetries to be detected without a preprocessing step; these symmetries are solely detected within the `GeneralRemoveAsymmetry` procedure.

Proposition 5. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_5-symmetric (resp. T_6-symmetric) in the pair (x_i, x_j) and $i < j$ iff every ROBDD rooted at a node labeled x_i is T_5-symmetric (resp. T_6-symmetric) in (x_i, x_j).*

Proposition 6. *An ROBDD f over a set of variables $\{x_1, \ldots, x_n\}$ is T_7-symmetric (resp. T_8-symmetric) in the pair (x_i, x_j) and $i < j$ iff*

1) *every ROBDD rooted at a node labeled x_i is T_7-symmetric (resp. T_8-symmetric) in (x_i, x_j) and,*
2) *every path from the root of f to a node h labeled x_j which does not visit a node labeled x_i, satisfies the property that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$.*

Proposition 7. *An ROBDD f over a set of variables $\{x_1,\ldots,x_n\}$ is T_{11}-symmetric (resp. T_{12}-symmetric) in the pair (x_i,x_j) and $i<j$ iff*

1) *every ROBDD rooted at a node labeled x_i is T_{11}-symmetric (resp. T_{12}-symmetric) in (x_i,x_j) and,*
2) *every path from the root of f passes through a node labeled x_i.*

The following two lemmata detail structural properties of ROBDDs that hold in the presence of T_5,T_6,T_7,T_8,T_{11} and T_{12}-symmetries. The absence of these properties imply that these symmetries cannot hold. In the case of Lemma 5, an $O(n|G|)$ complexity algorithm can be applied to ascertain whether every ROBDD rooted at a node labeled x_i contains a node labeled x_j. This result therefore provides a sieve for T_5 and T_6-symmetries that can be incorporated into `FindSlowAsymmetry`. A sieve for T_7,T_8,T_{11} and T_{12}-symmetries follows from Lemma 6 since the two cases of the lemma can both be checked in $O(n|G|)$ time. This is also implemented within `FindSlowAsymmetry`.

Lemma 5. *If an ROBDD f over a set of variables $\{x_1,\ldots,x_n\}$ is T_5-symmetric (resp. T_6-symmetric) in the pair (x_i,x_j) and $i<j$ then every ROBDD rooted at a node labeled x_i contains a node labeled x_j.*

Lemma 6. *If an ROBDD f over a set of variables $\{x_1,\ldots,x_n\}$ is T_7-symmetric (resp. T_8-symmetric, T_{11}-symmetric and T_{12}-symmetric) in the pair (x_i,x_j) and $i<j$ then every ROBDD g rooted at a node labeled x_i satisfies the property that*

1) *g contains a node labeled x_j or,*
2) *$g|_{x_i\leftarrow 0}=\neg g|_{x_i\leftarrow 1}$.*

The recursive structure of `GeneralRemoveAsymmetry` follows that of `RemoveAsymmetry` except that the call `GeneralRemoveAsymmetryVar`$(f|_{x_i\leftarrow 0}, f|_{x_i\leftarrow 1}, C)$ lies at its heart. `GeneralRemoveAsymmetryVar` in turn mimics the structure of `RemoveAsymmetryVar` except that it performs co-factor checks for $T_1,T_2,T_5,T_6,T_7,T_8,T_{11}$ and T_{12}-symmetries. Note that the T_3,T_4,T_9 and T_{10}-symmetries are already completely determined by `FindFastSymmetry` and hence need not be reconsidered. The complexity of a single call to `GeneralRemoveAsymmetryVar` is $O(|G|^2)$ and since this function can only be invoked a total of $|G|$ times in Algorithm 5 when caching is applied, it follows that the overall complexity of this procedure is $O(|G|^3)$. The preprocessing checks implemented within `FindSlowAsymmetry` for Propositions 2, 4 and 7 all require $O(n|G|)$ time whereas the preprocessing required for Propositions 3 and 6 take $O(n^2|G|)$. Algorithm 5 thus resides in $O(n^2|G|+|G|^3)$ overall.

C. Generalized Symmetry Propagation

To reduce the cost of each iteration of the main loop of Algorithm 5, one can apply asymmetry/symmetry propagation in the spirit of that employed in Algorithm 4. Tsai *et al.* [22] have reported transitivity results for some generalized symmetries, but to fully exploit asymmetry/symmetry propagation these results need to be extended to all 12 generalized symmetries.

Algorithm 6 GeneralRemoveAsymmetry(f, i, C)

> **if** $C=\emptyset \vee f=\mathbf{true} \vee f=\mathbf{false}$ **then**
>   **return** C
> $j \leftarrow$ `index`(f)
> **if** $j>i$ **then**
>   **return** C
> **else if** $j=i$ **then**
>   **return** `GeneralRemoveAsymmetryVar`$(f|_{x_i\leftarrow 0}, f|_{x_i\leftarrow 1}, C)$
> **else**
> $C \leftarrow$ `GeneralRemoveAsymmetry`$(f|_{x_j\leftarrow 0}, i, C)$
>   **return** `GeneralRemoveAsymmetry`$(f|_{x_j\leftarrow 1}, i, C)$

Algorithm 7 GeneralRemoveAsymmetryVar(g_0, g_1, C)

> **if** $g_0=\mathbf{true} \vee g_0=\mathbf{false}$ **then**
> $j \leftarrow \infty$
> **else**
> $j \leftarrow$ `index`(g_0)
> **if** $g_1=\mathbf{true} \vee g_1=\mathbf{false}$ **then**
> $r \leftarrow \infty$
> **else**
> $r \leftarrow$ `index`(g_1)
> **if** $C=\emptyset \vee j=r=\infty$ **then**
>   **return** C
> **else if** $j=r$ **then**
> $(l,g_{00},g_{01},g_{10},g_{11}) \leftarrow (j, g_0|_{x_j\leftarrow 0}, g_0|_{x_j\leftarrow 1}, g_1|_{x_r\leftarrow 0}, g_1|_{x_r\leftarrow 1})$
> **else if** $j<r$ **then**
> $(l,g_{00},g_{01},g_{10},g_{11}) \leftarrow (j, g_0|_{x_j\leftarrow 0}, g_0|_{x_j\leftarrow 1}, g_1, g_1)$
> **else**
> $(l,g_{00},g_{01},g_{10},g_{11}) \leftarrow (r, g_0, g_0, g_1|_{x_r\leftarrow 0}, g_1|_{x_r\leftarrow 1})$
> **if** $g_{10}\neq g_{01}$ **then**
> $C \leftarrow C \setminus \{(l,1)\}$
> **if** $g_{00}\neq g_{11}$ **then**
> $C \leftarrow C \setminus \{(l,2)\}$
> **if** $g_{00}\neq g_{10}$ **then**
> $C \leftarrow C \setminus \{(l,5)\}$
> **if** $g_{01}\neq g_{11}$ **then**
> $C \leftarrow C \setminus \{(l,6)\}$
> **if** $g_{10}\neq \neg g_{01}$ **then**
> $C \leftarrow C \setminus \{(l,7)\}$
> **if** $g_{00}\neq \neg g_{11}$ **then**
> $C \leftarrow C \setminus \{(l,8)\}$
> **if** $g_{00}\neq \neg g_{10}$ **then**
> $C \leftarrow C \setminus \{(l,11)\}$
> **if** $g_{01}\neq \neg g_{11}$ **then**
> $C \leftarrow C \setminus \{(l,12)\}$
> $C \leftarrow$ `GeneralRemoveAsymmetryVar`(g_{00}, g_{10}, C)
> **return** `GeneralRemoveAsymmetryVar`(g_{01}, g_{11}, C)

One such extension that involves T_1 and T_3-symmetries is presented in the following lemma:

Lemma 7. *If a Boolean function f over a set of variables $\{x_1,\ldots,x_n\}$ is T_1-symmetric in the pair (x_i,x_j) and T_3-symmetric in the pair (x_j,x_k), then f is T_3-symmetric in the pair (x_i,x_k).*

Proof. Suppose $T_1^{x_i,x_j}(f)$ and $T_3^{x_j,x_k}(f)$ hold. Thus $f|_{x_i\leftarrow 1,x_j\leftarrow 0} = f|_{x_i\leftarrow 0,x_j\leftarrow 1}$, therefore $f|_{x_i\leftarrow 1,x_j\leftarrow 0,x_k\leftarrow 0} = f|_{x_i\leftarrow 0,x_j\leftarrow 1,x_k\leftarrow 0}$ and likewise $f|_{x_i\leftarrow 1,x_j\leftarrow 0,x_k\leftarrow 1} = f|_{x_i\leftarrow 0,x_j\leftarrow 1,x_k\leftarrow 1}$. Also $f|_{x_j\leftarrow 0,x_k\leftarrow 0} = f|_{x_j\leftarrow 0,x_k\leftarrow 1}$, thus $f|_{x_i\leftarrow 0,x_j\leftarrow 0,x_k\leftarrow 0} = f|_{x_i\leftarrow 0,x_j\leftarrow 0,x_k\leftarrow 1}$ and $f|_{x_i\leftarrow 1,x_j\leftarrow 0,x_k\leftarrow 0} = f|_{x_i\leftarrow 1,x_j\leftarrow 0,x_k\leftarrow 1}$. Therefore $f|_{x_i\leftarrow 0,x_j\leftarrow 0,x_k\leftarrow 0} = f|_{x_i\leftarrow 0,x_j\leftarrow 0,x_k\leftarrow 1}$ and $f|_{x_i\leftarrow 0,x_j\leftarrow 1,x_k\leftarrow 0} = f|_{x_i\leftarrow 1,x_j\leftarrow 0,x_k\leftarrow 0} = f|_{x_i\leftarrow 1,x_j\leftarrow 0,x_k\leftarrow 1} = f|_{x_i\leftarrow 0,x_j\leftarrow 1,x_k\leftarrow 1}$. Hence $f|_{x_i\leftarrow 0,x_k\leftarrow 0}$

II: Transitivity Results

	$T_1^{y,z}$	$T_7^{y,z}$	$T_2^{y,z}$	$T_8^{y,z}$	$T_3^{y,z}$	$T_9^{y,z}$	$T_4^{y,z}$	$T_{10}^{y,z}$	$T_5^{y,z}$	$T_{11}^{y,z}$	$T_6^{y,z}$	$T_{12}^{y,z}$
$T_1^{x,y}$	$T_1^{x,z}$ †	$T_7^{x,z}$ †	$T_2^{x,z}$ †	$T_8^{x,z}$ †	$T_3^{x,z}$	$T_9^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_5^{x,z}$	$T_{11}^{x,z}$	$T_6^{x,z}$	$T_{12}^{x,z}$
$T_7^{x,y}$	$T_7^{x,z}$ †	$T_1^{x,z}$ †	$T_8^{x,z}$ †	$T_2^{x,z}$ †	$T_3^{x,z}$	$T_9^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_{11}^{x,z}$	$T_5^{x,z}$	$T_{12}^{x,z}$	$T_6^{x,z}$
$T_2^{x,y}$	$T_2^{x,z}$ †	$T_8^{x,z}$ †	$T_1^{x,z}$ †	$T_7^{x,z}$ †	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$	$T_5^{x,z}$	$T_{11}^{x,z}$	$T_6^{x,z}$	$T_{12}^{x,z}$
$T_8^{x,y}$	$T_8^{x,z}$ †	$T_2^{x,z}$ †	$T_7^{x,z}$ †	$T_1^{x,z}$ †	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$	$T_{11}^{x,z}$	$T_5^{x,z}$	$T_{12}^{x,z}$	$T_6^{x,z}$
$T_3^{x,y}$	$T_3^{x,z}$	$T_9^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$				
$T_9^{x,y}$	$T_9^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$	$T_3^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$	$T_3^{x,z}$	$T_9^{x,z}$				
$T_4^{x,y}$	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$				
$T_{10}^{x,y}$	$T_{10}^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_4^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$	$T_4^{x,z}$	$T_{10}^{x,z}$				
$T_5^{x,y}$	$T_5^{x,z}$	$T_5^{x,z}$	$T_6^{x,z}$	$T_6^{x,z}$					$T_5^{x,z}$	$T_5^{x,z}$	$T_6^{x,z}$	$T_6^{x,z}$
$T_{11}^{x,y}$	$T_{11}^{x,z}$	$T_{11}^{x,z}$	$T_{12}^{x,z}$	$T_{12}^{x,z}$					$T_{11}^{x,z}$	$T_{11}^{x,z}$	$T_{12}^{x,z}$	$T_{12}^{x,z}$
$T_6^{x,y}$	$T_6^{x,z}$	$T_6^{x,z}$	$T_5^{x,z}$	$T_5^{x,z}$					$T_5^{x,z}$	$T_5^{x,z}$	$T_6^{x,z}$	$T_6^{x,z}$
$T_{12}^{x,y}$	$T_{12}^{x,z}$	$T_{12}^{x,z}$	$T_{11}^{x,z}$	$T_{11}^{x,z}$					$T_{11}^{x,z}$	$T_{11}^{x,z}$	$T_{12}^{x,z}$	$T_{12}^{x,z}$

$= f|_{x_i \leftarrow 0, x_k \leftarrow 1}$ and $T_3^{x_i,x_k}(f)$ holds. □

Table II summarizes a collection of lemmata that state implicational relationships between various generalized symmetries. For example, if $T_3^{x_i,x_j}(f)$ and $T_4^{x_j,x_k}(f)$ hold for some ROBDD f then $T_3^{x_i,x_k}(f)$ also holds. Implicational relationships that have been previously reported [22] are marked with a †. Proofs for all the other implicational relationships of Table II can be found in the accompanying technical report [34]. Many of these results are established with proofs whose structure mirrors that used to substantiate lemma 7. The correctness of the remaining results, flows from multiple applications of the following lemma that states equivalences between the generalized symmetries of the form $T_i^{x,y}(f)$ and $T_j^{y,x}(f)$ for any ROBDD f for various $i, j \in \{1, \ldots, 12\}$.

Lemma 8.

1) $T_1^{x,y}(f) \iff T_1^{y,x}(f)$ and $T_7^{x,y}(f) \iff T_7^{y,x}(f)$
2) $T_2^{x,y}(f) \iff T_2^{y,x}(f)$ and $T_8^{x,y}(f) \iff T_8^{y,x}(f)$
3) $T_3^{x,y}(f) \iff T_5^{y,x}(f)$ and $T_9^{x,y}(f) \iff T_{11}^{y,x}(f)$
4) $T_4^{x,y}(f) \iff T_6^{y,x}(f)$ and $T_{10}^{x,y}(f) \iff T_{12}^{y,x}(f)$

Proof. For brevity we only consider the positive cases.

- $T_1^{x,y}(f) \iff f|_{x \leftarrow 1, y \leftarrow 0} = f|_{x \leftarrow 0, y \leftarrow 1} \iff f|_{y \leftarrow 1, x \leftarrow 0} = f|_{y \leftarrow 0, x \leftarrow 1} \iff T_1^{y,x}(f)$
- $T_2^{x,y}(f) \iff f|_{x \leftarrow 0, y \leftarrow 0} = f|_{x \leftarrow 1, y \leftarrow 1} \iff f|_{y \leftarrow 0, x \leftarrow 0} = f|_{y \leftarrow 1, x \leftarrow 1} \iff T_2^{y,x}(f)$
- $T_3^{x,y}(f) \iff f|_{x \leftarrow 0, y \leftarrow 0} = f|_{x \leftarrow 0, y \leftarrow 1} \iff f|_{y \leftarrow 0, x \leftarrow 0} = f|_{y \leftarrow 1, x \leftarrow 0} \iff T_5^{y,x}(f)$
- $T_4^{x,y}(f) \iff f|_{x \leftarrow 1, y \leftarrow 0} = f|_{x \leftarrow 1, y \leftarrow 1} \iff f|_{y \leftarrow 0, x \leftarrow 1} = f|_{y \leftarrow 1, x \leftarrow 1} \iff T_6^{y,x}(f)$

□

The value of the above lemma is that it can be applied to show, for example, that the $T_3^{x,y}/T_7^{y,z}$ entry of Table II is a consequence of the $T_7^{x,y}/T_5^{y,z}$ entry. In fact three applications of the above lemma are needed to establish the correctness of the $T_3^{x,y}/T_7^{y,z}$ entry, as formalised in the following lemma.

Lemma 9. *If a Boolean function f over a set of variables $\{x_1, \ldots, x_n\}$ is T_3-symmetric in the pair (x, y) and T_7-symmetric in the pair (y, z), then f is T_9-symmetric in the pair (x, z).*

Proof. Suppose $T_3^{x,y}(f)$ and $T_7^{y,z}(f)$ hold. By two applications of Lemma 8 it follows that $T_5^{y,x}(f)$ and $T_7^{z,y}(f)$ hold. Hence $T_7^{z,y}(f)$ and $T_5^{y,x}(f)$ hold. By Table II it follows that $T_{11}^{z,x}(f)$ holds and by another application of Lemma 8 it follows that $T_9^{x,z}(f)$ holds as required. □

We conjecture that no implicational symmetry relationships hold for the combinations of symmetry that lead to a blank entry in the table.

With the results of Table II in place, it is straightforward to construct an analogue of `SymmetryClosure`(A, S) for generalized symmetries. The complexity of the generalized closure algorithm remains $O(n^3)$, assuming that an incremental algorithm is applied. Thus the overall running time of generalized symmetry detection with asymmetry/symmetry propagation is $O(n^3 + n^2|G| + |G|^3)$.

VII. Experimental Results

The anytime algorithm and all its refinements have been implemented using the CUDD [35] Decision Diagram package, so as to assess the efficiency of the anytime approach. The rationale for this choice of package was that the Extra DD library [36], which implements Mishchenko's algorithm, also uses CUDD. The main experiments were performed on an UltraSPARC IIIi 900MHz based system, equipped with 16GB RAM, running the Solaris 9 Operating System, using `getrusage` to gauge CPU usage in seconds. The CUDD package, the Extra library, and our algorithm were all compiled with the GNU C Compiler version 3.3.0 with `-O3` enabled. The algorithms were run against a range of MCNC and ISCAS benchmark circuits of varying size [37], as well as several other benchmarks derived from the SAT literature. All timings are given in seconds and averaged over four runs.

Table III presents the results of these tests, the first four columns of the table give, respectively, the circuit name, number of input variables, number of defined functions (outputs) and the sum of the number of internal ROBDD nodes across all outputs (which does not consider sharing between outputs). Column |**S**| records the total number of all T_1-symmetric pairs found over all the outputs. Column **Read** gives the time in seconds to read in the benchmark circuit and construct the

III: T_1-symmetry Experimental Results with (above) and without (below) variable reordering applied

Circuit	# In	# Out	Σ|G|	|S|	Read	Naïve	Möller	Mish-GC	Mish+GC	Any	Sat	Adj	Close
alu2	10	6	192	4	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
alu4	14	8	1099	6	0.01	0.05	0.01	0.01	0.01	0.01	0.01	0.01	0.01
C1355	41	32	65323	0	5.62	49.95	31.93	0.02	0.09	2.13	1.67	1.68	1.68
C1908	33	25	17682	248	2.10	5.71	1.89	0.07	0.12	0.64	0.42	0.26	0.20
C2670	233	140	8904	1547	1.10	64.36	13.50	0.32	4.67	2.84	2.65	2.62	2.21
C3540	50	22	43334	81	14.00	38.37	0.99	0.94	6.84	3.45	2.89	2.35	1.99
C432	36	7	1475	0	0.16	0.64	0.03	0.02	0.02	0.02	0.01	0.01	0.01
C499	41	32	101701	0	3.00	77.09	55.86	0.04	0.09	2.62	2.41	2.42	2.44
C5315	178	123	9434	521	0.72	5.69	0.50	0.28	0.50	0.48	0.36	0.34	0.29
C7552	207	108	29142	1879	7.36	366.18	191.69	0.70	6.34	3.57	3.21	2.01	2.68
C880	60	26	8753	262	0.44	5.20	0.13	0.22	1.01	0.24	0.16	0.12	0.10
dalu	75	16	1728	982	0.45	1.05	0.10	0.06	0.08	0.13	0.11	0.08	0.07
des	256	245	6063	1264	0.35	0.43	0.21	0.13	0.16	0.16	0.15	0.12	0.10
frg2	143	139	2339	1353	0.11	0.25	0.07	0.04	0.08	0.08	0.08	0.05	0.04
i10	257	224	52811	3746	9.49	98.13	4.14	2.09	427.69	1.87	1.54	1.52	1.27
k2	256	245	3029	338	0.04	0.79	0.03	0.07	0.10	0.07	0.04	0.02	0.01
pair	173	137	8599	1910	0.60	2.71	0.50	0.18	0.62	0.48	0.36	0.32	0.28
rot	135	107	4132	364	0.28	2.60	0.10	0.11	0.26	0.39	0.34	0.29	0.23
s4863	153	104	75549	547	87.58	14.78	0.80	0.09	1.28	0.50	0.32	0.29	0.16
s9234.1	247	250	9376	3454	2.16	6.76	0.76	0.39	1.46	0.87	0.74	0.68	0.42
s38584.1	1464	1730	34833	15629	13.10	18.36	1.72	2.89	4.11	4.83	3.26	2.96	2.80
too_large	38	3	2312	17	0.15	1.15	0.04	0.04	0.20	0.03	0.02	0.01	0.01
simp12	117	1	292811	23	230.61	>7200	22.19	12.61	61.96	55.55	22.22	21.81	21.96
hom08	95	1	110160	16	128.91	>7200	4.39	4.18	134.31	17.48	4.70	4.74	4.50
ca016	107	1	90033	26	33.45	6444.37	2544.87	19.54	>7200	20.19	17.01	16.36	14.10
urquhart4_25	68	1	45008	27	23.21	3330.31	1070.31	4.57	>7200	6.94	6.37	6.31	6.23
rope_0006	61	1	11066	13	5.01	564.39	216.53	0.40	28.17	1.28	1.03	0.99	0.98
ferry10	116	1	3141	38	6.18	140.32	64.45	0.34	>7200	0.44	0.42	0.46	0.48
gripper12	129	1	17035	43	165.65	>7200	>7200	7.05	5365.41	35.35	34.89	34.80	36.32
C1355	41	32	110675	0	10.25	111.41	52.68	0.13	0.33	6.11	5.89	5.90	5.91
C1908	33	25	30832	248	0.16	14.95	4.21	0.13	0.30	1.01	1.00	0.98	0.38
C2670	233	140	9869047	1547	39.19	>7200	3854.76	907.71	>7200	187.10	161.23	156.32	124.86
C3540	50	22	4618194	81	21.80	>7200	122.09	132.72	5488.75	71.64	68.23	66.08	65.04
C432	36	7	32151	0	0.20	14.36	0.38	0.77	45.23	0.68	0.46	0.45	0.45
C499	41	32	110675	0	0.14	94.66	50.72	0.40	0.45	5.29	4.97	4.96	4.96
C880	60	26	600998	262	8.29	704.54	10.23	13.90	2242.11	7.75	6.84	5.63	5.20
dalu	75	16	5128	982	0.06	1.43	0.38	0.12	0.17	0.67	0.64	0.61	0.34
des	256	245	15209	1264	0.19	0.73	0.47	0.15	0.33	0.21	0.20	0.17	0.11
frg2	143	139	6679	1353	0.04	0.47	0.05	0.11	0.19	0.09	0.08	0.07	0.04
i10	257	224	150353	3746	0.61	1203.85	30.26	5.89	>7200	5.61	5.12	4.86	4.12
pair	173	137	118066	1910	0.20	132.46	4.45	6.62	35.50	2.37	2.18	2.16	2.08
rot	135	107	13565	364	0.10	12.72	0.31	0.32	4.50	0.61	0.31	0.30	0.22
s4863	153	104	126988	547	2.63	20.60	1.45	5.30	5.71	1.41	1.08	1.01	0.82
s9234.1	247	250	4434504	3454	20.14	>7200	1415.88	1407.20	>7200	183.84	158.36	145.94	141.26
s38584.1	1464	1730	150554	15629	3.70	337.59	23.01	16.70	132.16	3.12	3.04	3.01	2.80
simp12	117	1	758330	23	76.23	>7200	139.45	>7200	>7200	105.67	61.94	59.87	57.59
hom08	95	1	893312	16	56.48	>7200	466.21	135.79	>7200	67.79	54.99	50.89	49.00
ca016	107	1	861209	26	60.10	>7200	744.55	305.11	>7200	72.68	59.96	50.90	50.80
urquhart4_25	68	1	1736705	27	5.96	>7200	974.83	>7200	>7200	83.44	81.84	76.48	72.02
rope_0006	61	1	759039	13	3.14	>7200	225.23	657.74	>7200	35.78	30.76	30.64	30.68
ferry10	116	1	539419	38	88.08	>7200	2177.43	1866.62	>7200	70.34	69.84	54.19	53.42
gripper12	129	1	667877	43	50.95	>7200	2604.07	368.50	>7200	106.32	102.87	85.43	84.90

ROBDD. The remaining columns give the runtimes required to compute all T_1-symmetric and T_1-asymmetric pairs. The first of these, **Naïve**, is the naïve method which computes all co-factor pairs. (The results of this method were used to verify the correctness of all subsequent methods.) The second column, **Möller**, applies the sieves of Sections V-A and V-B to reduce the number of co-factor calculations. The third and fourth columns, **Mish-GC** and **Mish+GC**, are Mishchenko's implementation of his own algorithm [36] without and with garbage collection enabled. The fifth column, **Any**, is the unoptimized anytime algorithm presented in Section IV. The remaining three columns, **Sat**, **Adj** and **Close**, are the times with the optimizations of Sections V-A, V-B and V-C cumulatively enabled. The garbage generated by Mishchenko's implementation stems from its use of ZBDDs to represent sets. Enabling garbage collection has not impact on our algorithm.

The columns labeled **Sat**, **Adj** and **Close** of Table III suggest that all the optimizations to the basic anytime algorithm are worthwhile, though not essential. Interestingly, computing transitive closure is not prohibitively expensive even when implemented using the sub-optimal Floyd-Warshall algorithm. This is because this algorithm can be implemented efficiently and straightforwardly with three nested loops. This simplicity of this optimization suggests that it should be applied in conjunction with the naïve method [20]. The rows of the table above the double lines record the outcomes of the experiments when circuits are constructed using dynamic variable ordering. The so-called automatic variable ordering option provided by CUDD was applied using the default settings which periodically activates the sifting algorithm of Rudell [21]. The rows beneath the double lines repeat the experiments with variable reordering disabled. This leads to much larger ROBDDs and therefore constitutes a form of strength test for all algorithms. Those benchmarks not repeated in the bottom section of the table correspond to those circuits which are the same size, with and without variable reordering.

Table III can only be meaningfully interpreted in conjunction with asymptotic complexity results. Complexity results, such as the assertion that the basic anytime algorithm resides in $O(|G|^3)$ assuming $n \leq |G|$, are ultimately statements about scalability; such results predict how the running time of an algorithm will grow with the size of the input ROBDD. These statements have particular weight when combined with the experimental results of Table III that gauge the asymptotic constants. For instance, if the basic anytime terminates within an acceptable time for very large ROBDDs then (no matter whether the ROBDD has been created with or without sifting, and irrespective of the number of symmetries inferred), the algorithm will terminate within an acceptable time for smaller ROBDDs. This is because the total number of atomic operations is $O(|G|)$. Interestingly, the algorithm of Mishchenko is $O(|G|^3)$ in the number of set operations, where each set operation will have variable complexity depending, for instance, on the number of represented symmetry pairs. Moreover, when sets are realised as ZBDDs, the cost of each set operation will also vary due to memoization (caching) effects and the overheads induced by memory management. This variability is evident in the columns **Mish-GC** and **Mish+GC**. This key difference in the asymptotic complexity explains why, although the running time of the anytime algorithms are consistently below 200 secs, and certainly never exceeds 2 hours, that these algorithms are not uniformly faster than the algorithm of Mishchenko because of the variability of its ZBDD operations.

Table V presents a comparison between the generalized symmetry algorithm of Zhang *et al.* [12] and the generalized anytime approach. Mishchenko's implementation was modified to detect T_1, T_2, T_7 and T_8-symmetries following the ideas prescribed by Zhang *et al.* The timings given for the anytime algorithm reflect the time required to compute all 12 generalized symmetry types. This algorithm applies asymmetry/symmetry propagation between iterations of the main loop and uses all sieves described thus far.

Figure 2 summarizes the outcome of some experiments that investigate the relationship between the variable choice heuristics and the proportion of symmetries found early in the execution of the algorithm. The graphs display the number of symmetries found against various timeouts for the $\min$ and $\max$ heuristics using the original algorithm as a control. Apart from the circuits *hanoi4*, *homer08* and *rope_0006* (graphs 9, 10 and 11) the $\min$ heuristic increases the proportion of symmetries found early in the execution of the algorithm. In the case of *dp02s02* (graph 5) and *gripper12* (graph 8), the difference between $\min$ and both the control and $\max$ is stark. This suggests that the $\min$ heuristic should always be applied since it never gives a significant slowdown when the algorithm is run to completion and is beneficial in the case of early termination. For five of the circuits (graphs 6 to 10) the number of symmetries grows consistently with time. However, for other circuits, growth is either more sporadic or biased towards the latter passes of the symmetry detection algorithm. For these circuits, only a fraction of symmetry pairs could be recovered if these algorithms were terminated prematurely. This is why it is important that anytime generality should not be achieved at the expense of efficiency.

Finally, one may wonder how the performance of the classical and generalized anytime algorithms are affected by the underlying architecture. Table IV thus summarises the results of some timing experiments performed with Intel Core2 Duo 2.33GHZ PC (using just one core), equipped with 2GB of RAM, running MacOSX. The Intel is faster than the UltraSPARC, but the memory limit of 2GB prevents some circuits (including all those for the larger SAT benchmarks) from being constructed. The **Mish** and **Zhang** columns detail the timings for the algorithms of Mishchenko and Zhang where garbage collection is disenabled. As before, the running times of the ZBDDs algorithms is more variable than those of the anytime algorithms. It should be noted the relative timings of the algorithms may change even between Intel machines, due to different memory speeds and caching behaviour.

VIII. Discussion

This paper presents a class of novel anytime symmetry detection algorithms. The tractability of these algorithms stem from their use of a single static adjacency matrix to represent

IV: Generalized Symmetry Experimental Results

		with variable reordering			without variable reordering		
Circuit	**|S|**	**Naïve**	**Zhang-GC**	**Anytime**	**Naïve**	**Zhang-GC**	**Anytime**
alu2	29	0.01	0.01	0.01	0.01	0.01	0.01
alu4	35	0.05	0.01	0.01	0.07	0.01	0.01
C1908	2160	9.00	0.50	1.85	24.24	1.34	3.29
C2670	5805	106.96	1.33	2.96	>7200	1106.96	102.69
C3540	1892	72.74	5.47	5.43	>7200	162.91	186.32
C432	212	1.03	0.04	0.12	29.37	95.24	2.93
C499	256	136.53	5.52	16.50	169.79	1.45	16.93
C5315	12515	13.13	2.25	1.90	-	-	-
C7552	13010	801.86	12.72	22.49	-	-	-
C880	1759	9.67	0.62	1.13	1309.88	42.39	44.52
dalu	5010	1.65	0.19	0.22	2.49	1.18	1.30
des	8917	0.64	1.69	0.43	1.43	4.80	0.70
frg2	11556	0.40	0.41	0.19	1.00	0.98	0.30
i10	40511	174.88	27.72	19.81	1802.24	63.73	70.29
k2	4750	1.26	0.34	0.14	1.38	0.32	0.15
pair	15949	4.56	1.53	1.21	219.76	64.27	9.10
rot	5948	4.38	0.78	1.05	25.67	10.66	2.57
s635	18451	0.18	0.19	0.05	0.18	0.18	0.03
s838.1	18588	0.42	0.20	0.05	0.38	0.15	0.06
s1196	879	0.25	0.05	0.04	0.42	0.17	0.08
s1269	912	1.13	0.24	0.24	1.67	0.41	0.32
s1423	20947	6.88	1.19	1.10	30.01	2.90	1.81
s3271	3577	0.23	0.27	0.08	2.46	1.15	0.42
s4863	3825	25.25	14.20	4.36	33.10	15.20	5.42
s9234.1	22410	13.53	3.78	1.12	>7200	>7200	287.62
s38584.1	136537	30.44	246.37	2.59	501.34	576.39	10.30
too_large	502	2.03	0.21	0.15	1.87	0.39	0.15
simp12	135	>7200	70.33	202.89	>7200	>7200	304.21
hom08	108	>7200	71.44	113.58	>7200	482.30	281.57
ca016	147	>7200	198.45	10.78	>7200	305.11	72.68
urquhart4_25	184	>7200	>7200	67.70	>7200	>7200	83.44
rope_0006	76	781.21	17.20	14.93	>7200	657.74	35.78
ferry10	174	210.82	3050.82	3.91	>7200	3146.64	365.93
gripper12	220	>7200	59.98	247.64	>7200	673.09	587.28

pairs of symmetric variables. It is important to appreciate that there is no obvious way to re-engineer Mishchenko's algorithm to use a static adjacency matrix. This is because Mishchenko's algorithm is a bottom-up, divide-and-conquer algorithm that derives the solution to a problem by obtaining, and combining, the solutions to several sub-problems. Mishchenko [17, p 1590] points out that caching of the answers to these sub-problems is required to reduce the computational complexity from exponential to polynomial yet this requires multiple data structures to be maintained. By contrast, the anytime approach merely has to mark nodes as visited in any of the ROBDD traversals. This explains why anytime generality does not need to compromise efficiency.

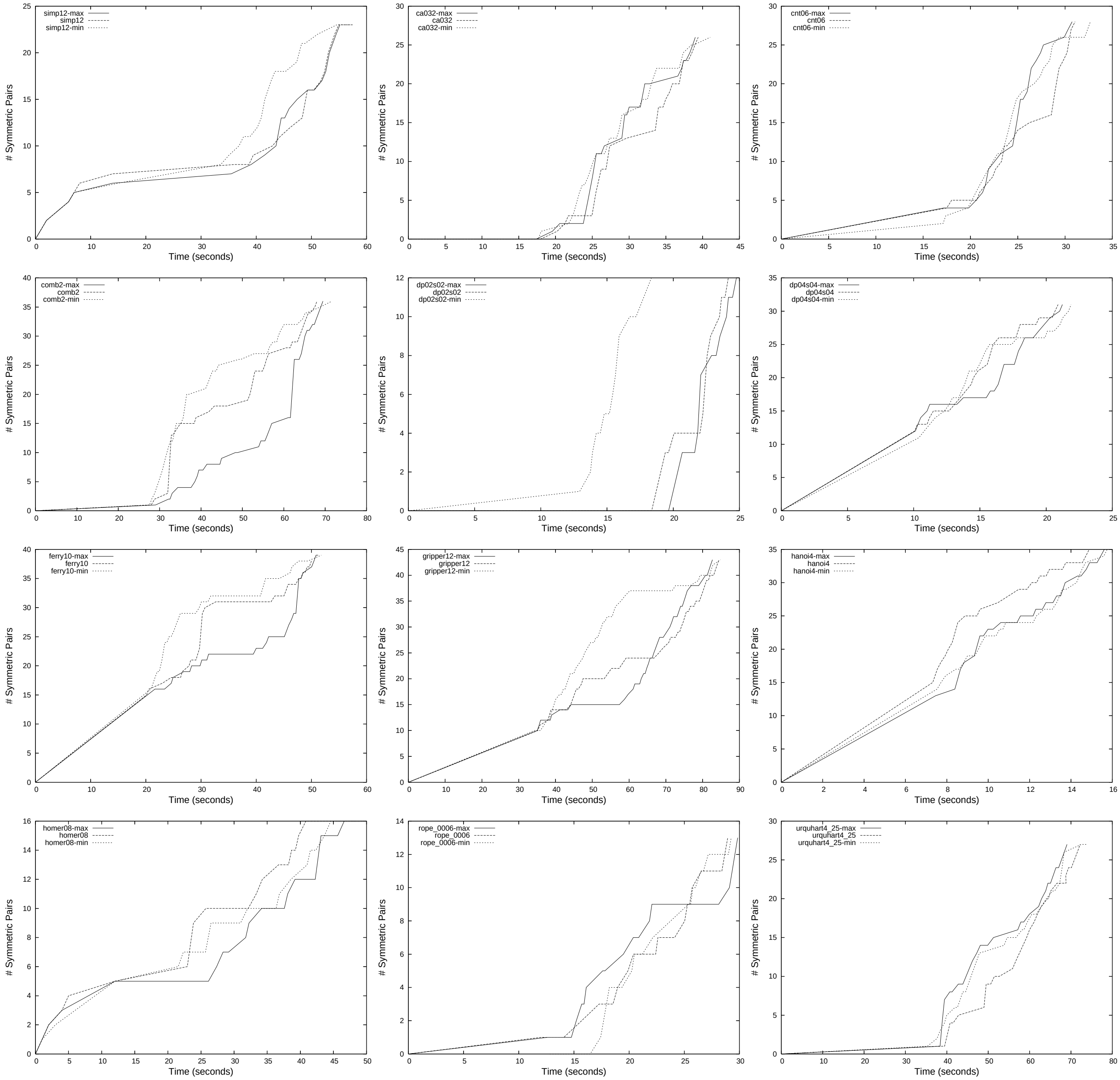
With a view to the future, the iterative nature of the anytime algorithms proposed in this paper make them good candidates for parallel evaluation on the 8 and 16 core processors that are predicated to emerge over the next 5 years. Although the speedups achieved by parallel evaluation of BDD operations have often been modest [38], the weak coupling between the iterations of the main loop of the symmetry detection algorithms — the property that yields to anytime execution — also leads to weakly coupled parallel execution.

ACKNOWLEDGMENTS

We thank Arnaud Gotlieb, Peter Schachte and Harald Søndergaard for discussions on ROBDDs, Jin Zhang for clarifying details of Mishchenko's algorithm, and the anonymous reviewers for their insightful comments and ideas.

REFERENCES

[1] C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *AIEE Trans.*, vol. 57, pp. 713–723, 1938.
[2] B. G. Kim and D. L. Dietmeyer, "Multilevel Logic Synthesis of Symmetric Switching Functions," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 4, pp. 436–446, 1991.
[3] C. R. Edward and S. L. Hurst, "A Digital Synthesis Procedure Under Function Symmetries and Mapping Methods," *IEEE Trans. Comput.*, vol. C-27, no. 11, pp. 985–997, 1978.



2: Symmetries against time

[4] F. Mailhot and G. De Micheli, "Technology Mapping Using Boolean Matching and Don't Care Sets," in *European Design Automation Conference*, 1990, pp. 212–216.

[5] Y. T. Lai, S. Sastry, and M. Pedram, "Boolean Matching Using Binary Decision Diagrams with Applications to Logic Synthesis and Verification," in *International Conference on Computer-Aided Design*, 1992, pp. 452–458.

[6] V. N. Kravets and K. A. Sakallah, "Constructive Library-Aware Synthesis using Symmetries," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2000, pp. 208–215.

[7] J. S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch, "Detecting Support-Reducing Bound Sets using Two-Cofactor Symmetries," in *Asia and South Pacific Design Automation Conference*, 2005, pp. 266–271.

[8] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams," in *International Conference on Computer-Aided Design*, 1994, pp. 628–631.

[9] C. Scholl, D. Möller, P. Molitor, and R. Drechsler, "BDD Minimization Using Symmetries," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 2, pp. 81–100, 1999.

[10] J. Mohnke and S. Malik, "Permutation and Phase Independent Boolean Comparison," *INTEGRATION, The VLSI Journal*, vol. 16, pp. 109–129, 1993.

[11] D. I. Cheng and M. Marek Sadowska, "Verifying Equivalence of Functions with Unknown Input Correspondence," in *European Design Automation Conference*, 1993, pp. 272–277.

[12] J. S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch, "Generalized Symmetries in Boolean Functions: Fast Computation and Application to Boolean Matching," in *Proceedings of the International Workshop on Logic Synthesis*, 2004, pp. 424–430.

[13] V. N. Kravets and K. A. Sakallah, "Generalized Symmetries in Boolean Functions," in *International Conference on Computer-Aided Design*,

V: Classical and Generalized Symmetry Timing Experiments on an Intel

	Classical								Generalized					
	with reordering				without reordering				with reordering			without reordering		
Circuit	**Naïve**	**Möller**	**Mish**	**Close**	**Naïve**	**Möller**	**Mish**	**Close**	**Naïve**	**Zhang**	**Close**	**Naïve**	**Zhang**	**Close**
alu2	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
alu4	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
C1908	2.53	0.73	0.04	0.16	6.69	2.10	0.12	1.12	3.75	0.17	1.23	5.50	0.29	1.16
C2670	35.71	5.57	0.12	0.17	–	–	–	–	53.46	2.58	2.62	–	–	–
C3540	13.34	0.29	0.35	0.24	–	–	–	–	24.37	3.29	1.59	–	–	–
C432	0.23	0.01	0.01	0.01	10.08	0.14	1.89	0.13	0.35	0.01	0.02	10.23	15.70	0.88
C499	41.80	35.14	0.04	0.98	62.04	30.10	0.10	1.12	32.18	0.08	1.02	40.80	0.33	3.23
C5315	3.12	0.17	0.07	0.11	–	–	–	–	4.71	0.40	0.62	–	–	–
C880	2.35	0.05	0.62	0.03	273.30	3.55	76.41	1.93	2.52	0.35	0.34	392.50	628.10	19.35
dalu	0.40	0.04	0.01	0.01	0.63	0.10	0.05	0.07	0.58	0.05	0.06	0.77	0.14	0.32
des	0.17	0.07	0.03	0.04	0.35	0.15	0.06	0.06	0.24	0.19	0.15	0.44	0.60	0.25
frg2	0.10	0.02	0.01	0.01	0.24	0.03	0.04	0.02	0.13	0.08	0.07	0.27	0.19	0.11
i10	42.85	1.43	1.36	0.56	410.74	21.13	77.45	1.96	58.68	115.71	6.96	>7200	4556.79	20.81
k2	0.37	0.01	0.03	0.01	0.36	0.02	0.04	0.01	0.50	0.08	0.05	0.50	0.08	0.05
pair	1.05	0.18	0.12	0.08	46.22	1.47	5.01	0.51	1.67	0.38	0.40	73.58	15.54	2.96
rot	1.05	0.03	0.03	0.03	6.68	0.13	0.12	0.07	1.70	0.18	0.23	9.02	2.40	0.86
s635	0.03	0.02	0.04	0.01	0.04	0.02	0.05	0.01	0.05	0.04	0.04	0.05	0.04	0.04
s838.1	0.07	0.02	0.04	0.01	0.08	0.02	0.05	0.01	0.10	0.04	0.05	0.12	0.04	0.05
s1196	0.05	0.01	0.02	0.01	0.11	0.01	0.02	0.01	0.07	0.03	0.02	0.15	0.04	0.03
s1269	0.27	0.02	0.02	0.01	0.45	0.03	0.03	0.01	0.39	0.07	0.06	0.60	0.10	0.10
s1423	1.26	0.10	0.07	0.08	6.86	0.58	0.15	0.13	1.60	0.21	0.30	8.96	0.80	0.81
s3271	0.03	0.01	0.01	0.01	0.59	0.12	0.05	0.03	0.06	0.05	0.04	0.82	0.18	0.13
s4863	5.50	0.35	0.02	0.07	9.84	0.48	0.04	0.21	8.45	0.41	0.96	17.48	1.23	1.75
s9234.1	1.51	0.20	0.10	0.08	–	–	–	–	1.89	0.50	0.51	–	–	–
too_large	0.38	0.01	0.02	0.01	0.47	0.01	0.02	0.01	0.58	0.10	0.06	0.56	0.09	0.03

2000, pp. 526–532.

[14] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[15] M. Chrzanowska-Jeske, "Generalized Symmetric Variables," in *International Conference on Electronics, Circuits, and Systems*, vol. 3, 2001, pp. 1147–1150.

[16] N. Kettle and A. King, "An Anytime Symmetry Detection Algorithm for ROBDDs," in *Asia and South Pacific Design Automation Conference*, 2006, pp. 243–248.

[17] A. Mishchenko, "Fast Computation of Symmetries in Boolean Functions," *IEEE Trans. Computer-Aided Design*, vol. 22, no. 11, pp. 1588–1593, 2003.

[18] B. Bollig and I. Wegener, "Improving the Variable Ordering of OBDDs is NP-complete," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, 1996.

[19] D. Sieling, "The Nonapproximability of OBDD Minimization," *Information and Computation*, vol. 172, no. 2, pp. 103–138, 2002.

[20] D. Möller, J. Mohnke, and M. Weber, "Detection of Symmetry of Boolean functions Represented by ROBDDs," in *International Conference on Computer-Aided Design*, 1993, pp. 680–684.

[21] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," in *International Conference on Computer-Aided Design*, 1993, pp. 42–47.

[22] C. C. Tsai and M. Marek-Sadowska, "Boolean Functions Classification via Fixed Polarity Reed-Muller Forms," *IEEE Trans. Comput.*, vol. 46, no. 2, pp. 173–186, 1997.

[23] J. S. Zhang, A. Mishchenko, R. Brayton, and M. Chrzanowska-Jeske, "Symmetry Detection for Large Boolean Functions using Circuit Representation, Simulation, and Satisfiability," in *Design Automation Conference*, 2006, pp. 510–515.

[24] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, 1986.

[25] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," in *Design Automation Conference*, 1993, pp. 272–277.

[26] G. Wang, A. Kuehlmann, and A. L. Sangiovanni-Vincentelli, "Structural Detection of Symmetries in Boolean Functions," in *International Conference on Computer Design*, 2003, pp. 498–503.

[27] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," in *Design Automation Conference*, 1990, pp. 40–45.

[28] P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov, "Exploiting Structure in Symmetry Detection for CNF," in *Design Automation Conference*, 2004, pp. 530–534.

[29] R. W. Floyd, "Algorithm 97: Shortest Path," *Commun. ACM*, vol. 5, no. 6, p. 345, 1962.

[30] S. Warshall, "A Theorem on Boolean Matrices," *Journal of the ACM*, vol. 9, no. 1, pp. 11–12, 1962.

[31] T. Ibaraki and N. Katoh, "On-line Computation of Transitive Closure of Graphs," *Information Processing Letters*, vol. 16, pp. 95–97, 1983.

[32] J. Mohnke, P. Molitor, and S. Malik, "Limits of Using Signatures for Permutation Independent Boolean Comparison," *Formal Methods in System Design*, vol. 21, no. 2, pp. 167–191, 2002.

[33] N. Kettle and A. King, "Proof of New Decompositional Results for Generalized Symmetries," University of Kent, Computing Laboratory, Tech. Rep. 05-06, 2006, http://www.cs.kent.ac.uk/pubs/2006/2402.

[34] ——, "Proof of New Implicational Relationships between Generalized Symmetries," University of Kent, Computing Laboratory, Tech. Rep. 13-05, 2006, http://www.cs.kent.ac.uk/pubs/2006/2349.

[35] F. Somenzi, "CUDD Package, Release 2.4.0." [Online]. Available: http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html

[36] A. Mishchenko, "Extra Library of DD Procedures." [Online]. Available: http://www.ee.pdx.edu/~alanmi/research/extra.htm

[37] "Lgsynth93 Benchmark Set." [Online]. Available: http://www.bdd-portal.org/benchmarks/

[38] K. Milvang-Jensen and A. J. Hu, "BDDNOW: A Parallel BDD Package," in *International Conference on Formal Methods in Computer-Aided Design*, ser. Lecture Notes in Computer Science, vol. 1522, 1998, pp. 501–507.